Swami Sahajanand College of Computer Science

B.C.A. SEM-V[NEP]

Subject: CORE JAVA
Major (Core) - 25617

UNIT 1

INTRODUCTION TO JAVA

- History of Java Features of Java, Applications of Java,
- Java Virtual Machine (JVM) and Byte Code, Buzz Words.
- Basics Concept of OOP: Abstraction and Encapsulation, Inheritance and Polymorphism
- Data types, Operators.
- Control Statement, Array, and command line argument.
- Structure of Java Programming.

B.C.A. Paper No: 504 – CORE JAVA **Credits: 04**

Mark Semester End Exam:
External Evaluation:50
Internal Evaluation:50 **Duration: 2.00Hrs**

Unit No	Course Contents	Teaching Hours	Weightage of Marks
Unit-1	**INTRODUCTION TO JAVA** • History of Java Features of Java, Applications of Java, Java Virtual Machine (JVM) and Byte Code, Buzz Words. • Basics Concept of OOP: Abstraction and Encapsulation, Inheritance and Polymorphism • Data types, Operators. • Control Statement, Array, and command line argument. • Structure of Java Programming.	15	25
Unit-2	**JAVA PROGRAMMING** • Classes, Objects and Methods. • Polymorphism: Method Overloading. • Constructor: Concept of Constructor, Types of Constructor, Constructor Overloading. • Garbage Collection • The 'this' keyword. 'static' and 'final' keyword. • Access Control: Public, Private, Protected, Default.	15	25
Unit-3	**INHERTANCE, INTERFACE AND PACKAGE** • Inheritance Basic, Types of Inheritance. • Method Overriding. • Run Time Polymorphism: Dynamic Method Dispatch. • Abstract Method and Class. • Interfaces: Defining Interface, Implementing Interface. • Implementation of Multiple and Hybrid Inheritance using Interface. Extending Interface	15	25
Unit-4	**PACKAGE AND EXCEPTION HANDLING** • Defining Package, Importing Packages. Access Protection • Built in Packages and user define package • Exception Handling Fundamentals, • Types of Exceptions. • Try...catch Keyword, Multiple Catch Statements. • Throw, Throws, Finally Keywords.	15	25

❖History of java, Basic OOP Concepts- Abstraction, Encapsulation, Inheritance, Polymorphism

Introduction to java

- Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique mission.
- Java has become inseparably linked with the online environment of the Internet
- Computer language innovation and development occurs for two fundamental reasons:
- To adapt to changing environments and uses
- To implement refinements and improvements in the art of programming
- The development of Java was driven by both elements in nearly equal measure.
- Java is related to C++, which is a direct descendant of C, Java is inherited from these two languages. From C, Java derives its syntax.
- Java was deeply rooted in the process of refinement and adaptation that has been occurringin computer programming languages for the past several decades.
- The development of Java was driven by both elements in nearly equal measure.

	JAVA	C
1	It is a pure Object Oriented Programming language.	It is Procedure Oriented Programming Language.
2	It does not include the 'C' unique statement, keywords goto, sizeof and typedef.	'C' includes all these keywords.
3	It does not contain the data type struct, union and enum.	It contains these data types.
4	It does not define the type modifiers keywords auto, extern, signed and unsigned.	It defines all these type of modifiers.
5	It does not support an explicit pointer type.	It supports explicit pointer type.
6	JAVA does not have a preprocessor and therefore we cannot use #include, #define and #ifdef statements.	It supports all these statements.
7	JAVA requires that the functions with no argument must be declared with empty parenthesis.	In 'C', you declared parenthesis with void keyword.
8	JAVA adds new operator such as instance of >>>(shift right with zero).	'C' does not support >>> operator.
9	Java support Inheritance, Encapsulation and Polymorphism.	'C' does not support all these features.
10	JAVA supports web base application.	'C' does not support web enable application.
11	It is developed by James Gosling.	It isdeveloped by Denis Ritche.

	JAVA	C++
1	JAVA is Pure (true) Object Oriented Programming Language.	C++ is basically 'C' with Object Oriented extension.
2	Java does not support operator overloading.	C++ supports operator overloading.
3	Java does not have template classes as in C++.	C++ have template classes.
4	Java does not support multiple inheritances of classes.	C++ supports multiple inheritances of classes.
5	JAVA adds new feature called "interface".	C++ does not support "interface".
6	JAVA does not use pointers.	C++ supports pointers.
7	There are no header files in JAVA.	C++ has a header files define first.
8	In JAVA, module operator (%) applies to the floating point types as well as integer types.	But in C++, only it apply to integer type only.
9	It is developed by James Gosling.	It is developed by BjarneStroustrup.

❖History of Java:

- Java is a general-purpose, object-oriented programming language developed by Sun Microsystem in 1991.
- It is originally called "Oak" by James Gosling, one of the inventors of the language.
- Java was designed for the development of software for consumer electronic devices like TV, VCR-DVD, Washing Machine, and Refrigerator.
- The Java team which included Patrick Naughton discovered that the existing language like C and C++ had limitations inn terms of both reliability and portability.
- They modelled their new language Java on C and C++ but removed some features of C and C++ that were considered as sources of problems.
- In 1990, Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
- In 1991, After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named "Oak".
- In 1992, The team, known as Green Project team by Sun,
- In 1993, The World Wide Web appeared on the Internet and transformed the text-based Internet into a graphical environment.
- In 1994, The team developed a Web browser called "HotJava" to locate and run applet programs on internet.
- In 1995, Oak was renamed "Java", due to some legal snags. Java is just a name and is not an acronym.
- In 1996, Java established itself not only as a leader for Internet programming but also general-purpose.
- In 1997, Sun release Java Development Kit 1.1 (JDK 1.1)
- In 1998, Sun releases Java 2 with version 1.2 of the Software Development Kit (SDK 1.2,

- In 1999, Sun releases Java 2 Platform, Standard Edition (J2SE) and Enterprise Edition (J2EE).
- In 2000, J2SE with SDK 1.3 was released.
- In 2002, J2SE with SDK 1.4 was released.
- In 2004, J2SE with JDK 5.0 was released.

❖OOP Concept:

What is OOPs?

- Programmingparadigm that views a computer program as a combination of objects which can exchangeinformation in a standardized manner, and can be combined with one another as modules or blocks.
- Each object is independent, can run by itself, and can be interlocked with other objects.
- Objects interact by passing information among each other, and each object contains information about itself and the objects it can interact with.
- Major OOP-oriented languages are C++, Java, and Smalltalk.
- Our definition of object oriented programming is : "Object oriented programming is an approach that provided a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.
- The major objective of object oriented approach is to eliminate some of the flaws encountered in the procedural approach.

Concepts:

Encapsulation:

- The wrapping up of data and methods into a single unit is known as encapsulation.
- Data encapsulation is the most striking feature of a class.
- The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.
- These methods provide the interface between the object's data the program.
- This insulation of the data form direct access by the program is called data hiding. Encapsulation makes it possible for objects to be treated like black box.

Abstraction:

-
- Abstraction is the act of representing essential features without including the background details or explanation.
- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and methods that operate on these attributes.

Inheritance:

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- Inheritance supports the concept of hierarchical classification.
- For example, the bird robin is a part of the class flying bird, which is again a part of the class bird.
- The concept of inheritance provides the idea of reusability.
- This means that we can add additional features to an existing one.
- The new class will have the combined features of both the classes.

Polymorphism:

- Polymorphism in another important OOP concept.
- Polymorphism means the ability to take more than one form.
- For example, an operation may exhibit different behavior in different case. The behavior depends upon the types of data used in the operation.
- For example, consider the operation of addition for two numbers, the operation will generate sum. If the operands are strings, then the operation would produce a third string by concatenation.
- Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface.
- This means that that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ.
- Polymorphism is extensively used in implementing inheritance.

❖Java Language Basics – Benefits, Applications, Byte Code, Buzzwords

Buzzwords/Benefits/Features of Java:

- The fundamental forces that necessitated the invention of Java are portability and security, other factors also played an important role in molding the final form of the language. They considerations were summed up by the Java team in the following list of buzzwords:

  • Simple
  • Secure
  • Portable
  • Object-oriented
  • Robust
  • Multithreaded
  • Architecture-neutral
  • Interpreted
  • High performance
  • Distributed
  • Dynamic

- Two of these buzzwords have already been discussed: secure and portable. Let's examine what each of the others implies.

Simple

- Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find Java hard to master. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier.

Secure.

- Java considers security as part of its design. The Java language, compiler, interpreter, and runtime environment were each developed with security in mind.

Platform-independent (portable)

- One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Object-Oriented

- Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model. The object model in Java is simple and easy to extend, while primitive types, such as integers, are kept as high-performance non objects.

Robust

- The multiplatform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java. To gain reliability, Java restricts you in a few key areas to force you to find your mistakes early in program development.

Multithreaded

- Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multi process synchronization that enables you to construct smoothly running interactive systems.

Architecture-Neutral

- A central issue for the Java designers was that of code longevity and portability. One of them in problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "write once; run anywhere, anytime, forever."

Interpreted and High Performance

- As described earlier, Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This code can be executed on any system that implements the Java Virtual Machine. Most previous attempts at cross-platform solutions have done so at the expense of performance. As explained earlier, the Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler. Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.

Distributed

- Java is designed for the distributed environment of the Internet because it handles TCP/IPprotocols. In fact, accessing a resource using a URL is not much different from accessing afile. Java also supports Remote Method Invocation (RMI). This feature enables a program toinvoke methods across a network.

Dynamic

- Java programs carry with them substantial amounts of run-time type information thatis used to verify and resolve accesses to objects at run time. This makes it possible todynamically link code in a safe and expedient manner. This is crucial to the robustness ofthe Java environment, in which small fragments of bytecode may be dynamically updatedon a running system.

Free

- People like FREE things, Don't you? So if a programmer wants to learn a programming language, or an organization wants to use a technology, COST is an important factor. Since Java is free from start, i.e. you don't need to pay anything to create Java application.

Everywhere

- Yes, Java is everywhere, it's on desktop, it's on mobile, it's on card, almost everywhere and so is Java programmers.

Applications of Java :

- There are many places where Java is used in real world, starting from commercial e-commerce website to android apps, from scientific application to financial applications like electronic trading systems, from games like Minecraft to desktop applications like Eclipse, Netbeans and IntelliJ, from open source library to J2ME apps etc.
- Let's see each of them in more detail.

Android Apps

- If you want to see where Java is used, you are not too far away. Open your Android phone and any app, they are actually written in Java programming language, with Google's Android API, which is similar to JDK.
- Couple of years back Android has provided much needed boost and today many Java programmer are Android App developer. By the way android uses different JVM and different packaging,

Server Apps at Financial Services Industry

- Java is very big in Financial Services. Lots of global Investment banks like Goldman Sachs, Citigroup, Barclays, Standard Charted and other banks use Java for writing front and back office electronic trading system, writing settlement and confirmation systems, data processing projects and several others. Java is mostly used to write server side application, mostly without any front end, which receives data form one server (upstream), process it and sends it other process (downstream). Java Swing was also popular for creating thick client GUIs for traders, but now C# is quickly gaining market share on that space and Swing is out of its breath.

Java Web applications

- Java is also big on E commerce and web application space. You have a lot of RESTfull services being created using Spring MVC, Struts 2.0 and similar frameworks. Even simple Servlet, JSP and Struts based web applications are quite popular on various government projects. Many of government, healthcare, insurance, education, defense and several other department have their web application built in Java.

Software Tools

- Many useful software and development tools are written and developed in Java e.g. Eclipse, InetelliJ Idea and Netbans IDE. I think they are also most used desktop applications written in Java. Though there was time when Swing was very popular to write thick client, mostly in financial service sector and Investment banks. Now days, Java FX is gaining popularity but still it is not a replacement of Swing and C# has almost replaced Swing in Finance domain.

Trading Application

- Third party trading application, which is also part of bigger financial services industry, also use Java. Popular trading application like Murex, which is used in many banks for front to bank connectivity, is also written in Java.

J2ME Apps

- Though advent of iOS and Android almost killed J2ME market, but still there is large market of low end Nokia and Samsung handset which uses J2ME. There was time when almost all games, application, which is available in Android are written using MIDP and CLDC, part of J2ME platform. J2ME is still popular on products like Blu-ray, Cards, Set top boxes etc. One of the reason of WhatsApp being so popular is because it is also available in J2ME for all those Nokia handset which is still quite big.

Embedded Space

- Java is also big in the embedded space. It shows how capable the platform is, you only need 130 KB to be able to use Java technology (on a smart card or sensor). Originally Java was designed for embedded devices. In fact, this is the one area, which was part of Java's initial campaign of "write once, run anywhere" and looks like it is paying up now.

Big Data technologies

- Hadoopand other big data technologies are also using Java in one way or other e.g. Apache's Java-based HBaseandAccumulo(open source), and ElasticSearchas well. By the Java is not dominating this space, as there are technologies like MongoDBwhich is written in C++. Java has potential to get major share on this growing space if HadooporElasticSearchgoes big.

High Frequency Trading Space

- Java platform has improved its performance characteristics a lot and with modern JITs, its capable of delivering performance at C++ level. Due to this reason, Java is also popular on writing high performance systems, because Though performance is little less compared to native language, but you can compromise safety, portability and maintainability for more speed and it only takes one inexperienced C++ programmer to make an application slow and unreliable.

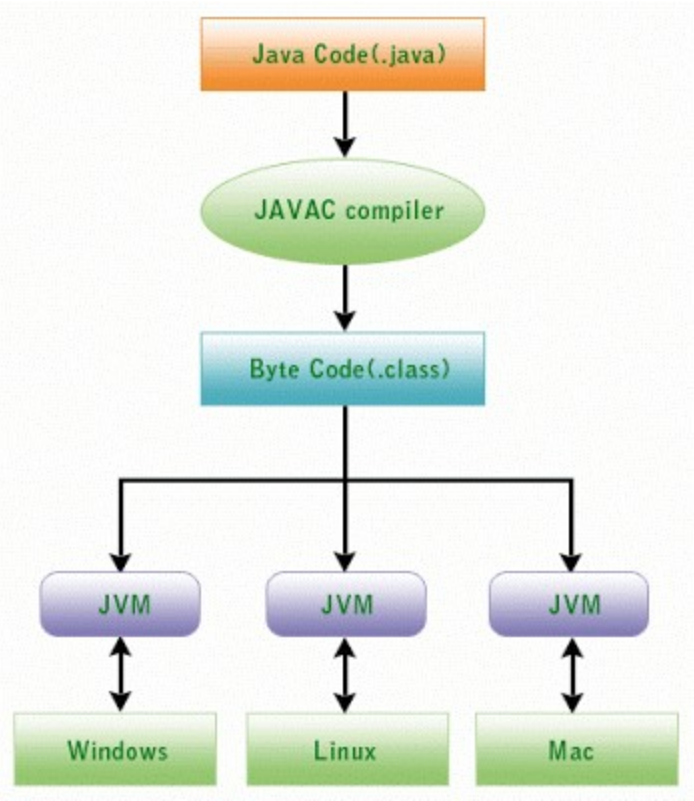
Scientific Applications

- Nowadays Java is often a default choice for scientific applications, including natural language processing. Main reason of this is because Java is more safe, portable, maintainable and comes with better high-level concurrency tools than C++ or any other language.

Bytecode:

- Bytecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code.
- This byte code is an machine independent code.

- It is not an completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by JVM.Bytecode is a machine code for JVM.
- But the machine code is platform specific whereas bytecode is platform independent that is the main difference between them.
- It is stored in .class file which is created after compiling the source code.
- Most developers although have never seen byte code.
- One way to view the byte code is to compile your class and then open the .class file in a hex editor and translate the bytecodes by referring to the virtual machine specification.
- A much easier way is to utilize the command-line utility javap. The Java SDK from Sun includes the javap disassembler, that will convert the byte codes into human-readable mnemonics.



- An example of bytecode is shown below
- Let us take a sample program.A Main class is created with login function

SOURCE CODE:

```
import java.io.*;
class Main
{
public static void main(String args[])
 {
new Login();
 }
}
```

- In command prompt compile it by typing "javac Main.java".
- After that type "javap -c Main".You will get the byte code as follows

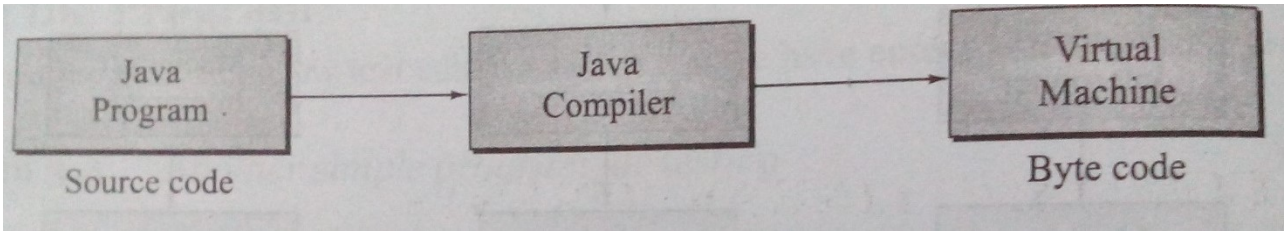
BYTE CODE:

```
Compiled from "Main.java"
class Main extends java.lang.Object{
Main();
  Code:
   0:  aload_0
   1:  invokespecial   #1; //Method java/lang/Object."":()V
   4:  return

public static void main(java.lang.String[]);
  Code:
   0:  new     #2; //class Login
   3:  dup
   4:  invokespecial   #3; //Method Login."":()V
   7:  pop
   8:  return

}
```

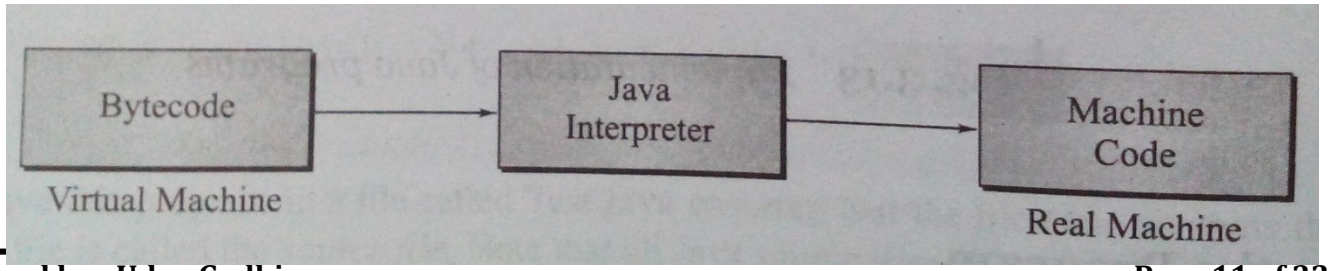
- The "javap -c" command disassembles a class file.
- Disassembling done by Disassembler which converts machine language to assembly language.

Java Virtual Machine (JVM);

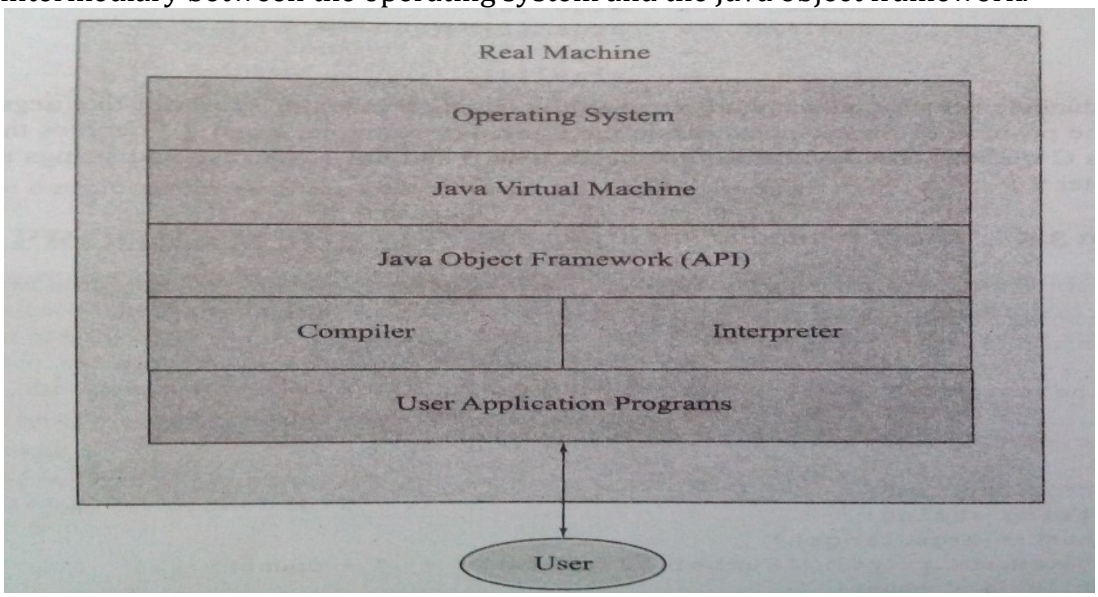
- All language compilers translate source code into machine code for a specific computer.
- Java compiler also does the same thing. Java compiler produces an intermedia code known as bytecode for a machine that does not exist.
- This machine is called Java Virtual Machine and it exists only inside the computer memory.
- It is simulated computer within the computer and does all major functions of a real computer.



- The virtual machine code is not machine specific. The machine specific doe is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in the following figure



- Remember that the interpreter is different for different machines.
- Java works on typical computer. The Java object framework (Java API) acts as the intermediary between the user program the user programs and the virtual machine which in turn acts as the intermediary between the operating system and the Java object framework.



Structure of java program, save, compile and run :

- Start Notepad, In a new document, type in the following code:
- /* The HelloWorldApp class implements an application that * displays "Hello World!" to the standard output. */

```
public class HelloWorldApp
{
        public static void main(String[] args)
        {
                System.out.println("Hello World!");
        }
}
```

- Save this code to a file. From the menu bar, select File > Save. In the Save As dialog box:
- Using the Save in drop-down menu, specify the folder (directory) where you'll save your file. In this example, the directory is java on the C drive. In the File name text box, type "HelloWorldApp.java", including the double quotation marks. From the Save as type drop-down menu, choose Text Document. When you're finished, the dialog box should look like this:

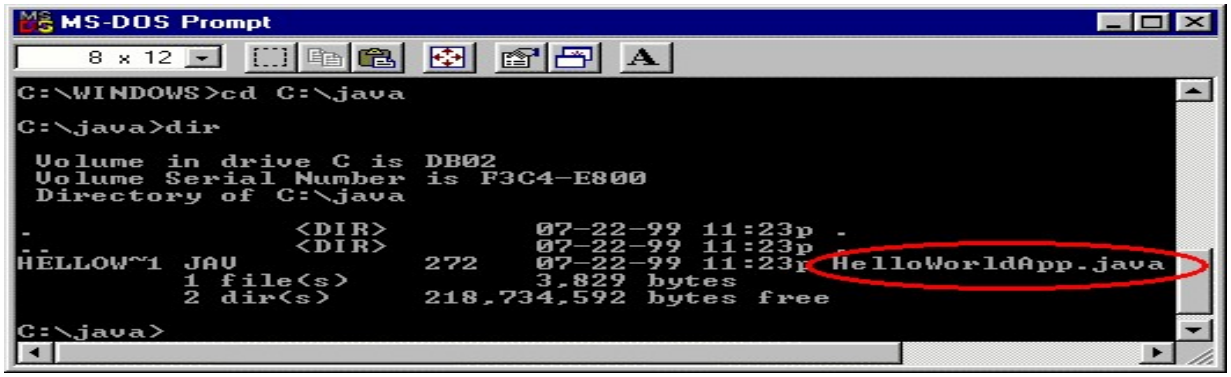
- Now click Save, and exit NotePad.

Compiling a program

- To Compile the Source File :

- From the Start menu, select the Command Prompt application (Windows XP). When the application launches, it should look like this:
- The prompt shows your current directory. When you bring up the prompt for Windows XP, your current directory is usually WINDOWS on your C drive. To compile your source code file, change your current directory to the directory where your file is located. For example, if your source directory is java on the C drive, you would type the following command at the prompt and pressEnter:

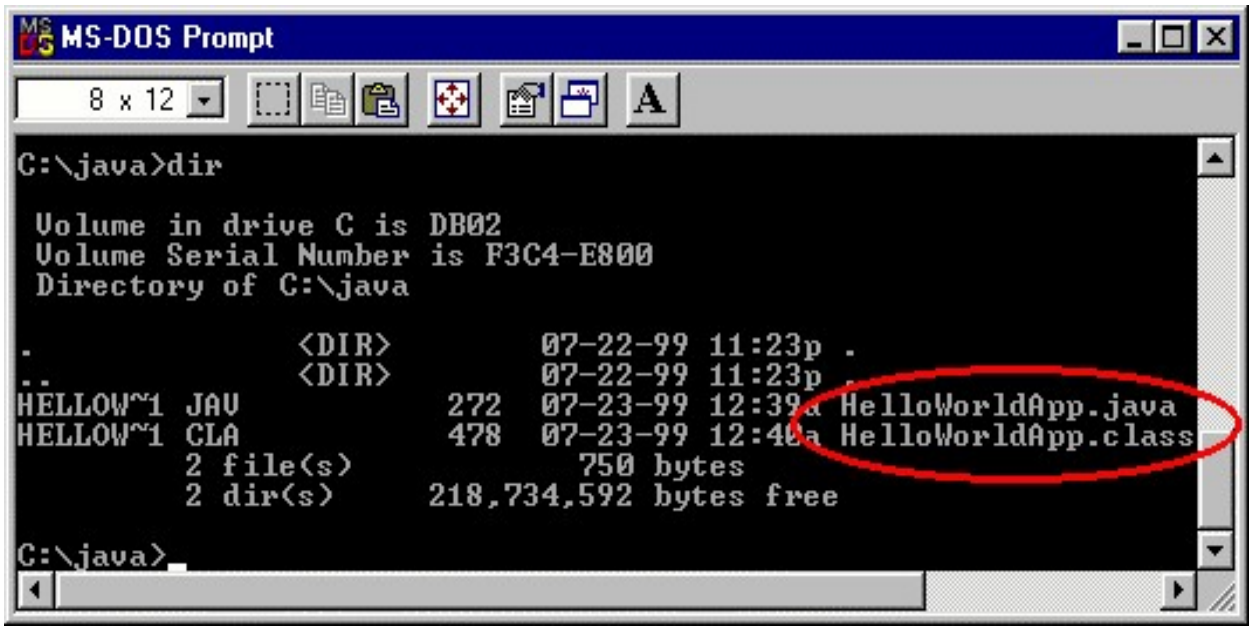
  cd c:\java
- Now the prompt should change to C:\java>.
- If you enter dir at the prompt, you should see your file.



- Now you can compile. At the prompt, type the following command and press Enter:

  javac HelloWorldApp.java

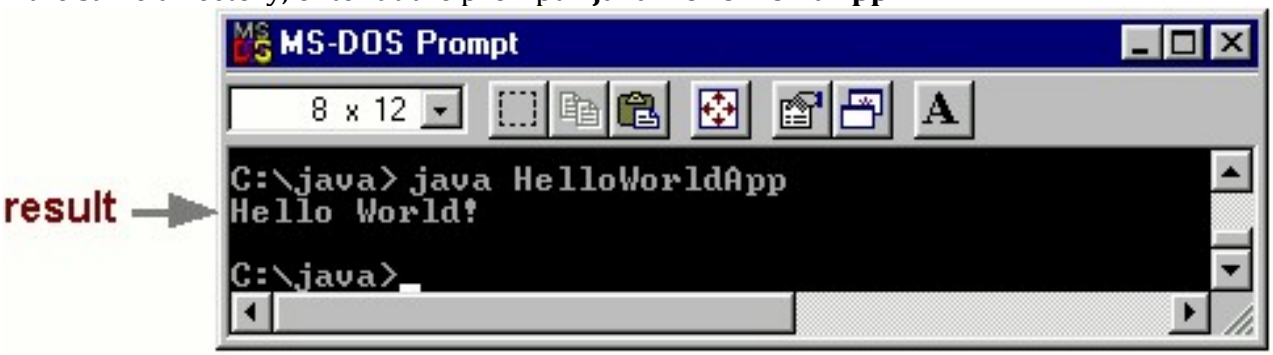
- The compiler has generated a Java bytecode file, HelloWorldApp.class. At the prompt, type dir to see the new file that was generated:



- Now that you have a .class file, so you can run your program.

Running a program :

- In the same directory, enter at the prompt: "**java HelloWorldApp**"



- Finally, you get the output : Hello World!

❖Data types, Variables, Arrays

Data types use in Java:

- You may remember two data types already, these being floating point (represented by the float keyword) and integer (represented by the int keyword).
- Java has eight different data types, all of which represent different kinds of values in a program.
- These data types are byte, short, int, long, float, double, char, and boolean.

(1) Integer Types

- The most common values used in computer programs are integers, which represent whole number values such as 12, 1988, and -34. Integer values can be both positive or negative, or even the value 0.
- The size of the value that's allowed depends on the integer data type you choose.
- Java features four integer data types, which are byte, short, int, and long.

Byte

- The first integer type, byte, takes up the least amount of space in a computer's memory.
- When you declare a constant or variable as byte, you are limited to values in the range -128 to 127.
- Why would you want to limit the size of a value in this way? Because the smaller the data type, the faster the computer can manipulate it.
- For example, your computer can move a byte value, which consumes only eight bits of memory, much faster than an int value, which, in Java, is four times as large.
- In Java, you declare a byte value like this:

```
byte identifier;
```

- In the preceding line, byte is the data type for the value, and identifier is the variable's name.
- You can also simultaneously declare and assign a value to a variable like this:

```
byte count = 100;
```

- After Java executes the preceding line, your program will have a variable named count that currently holds the value of 100.
- Of course, you can change the contents of count at any time in your program.

Short

- The next biggest type of Java integer is short. A variable declared as short can hold a value from -32,768 to 32,767.
- You declare a short value like this:

```
short identifier;
        or
short identifier = value;
```

- In the preceding line, value can be any value from -32,768 to 32,767, as described previously.
- In Java, short values are twice as big in memory-16 bits (or two bytes)-as byte values.

Int

- Next in the integer data types is int, which can hold a value from -2,147,483,648 to 2,147,483,647. Now you're getting into some big numbers.
- The int data type can hold such large numbers because it takes up 32 bits (four bytes) of computer memory.
- You declare int values like this:

```
int identifier;
```

or

int identifier = value;

Long

- The final integer data type in the Java language is long, which takes up a whopping 64 bits (eight bytes) of computer memory and can hold truly immense numbers.
- Unless you're calculating the number of molecules in the universe, you don't even have to know how big a long number can be.
- You declare a long value like this:

long identifier;

or

long identifier = value;

(2) Floating-Point Types

- Whereas integer values can hold only whole numbers, the floating-point data types can hold values with both whole number and fractional parts.
- Examples of floating-point values include 32.9, 123.284, and -43.436.
- As you can see, just like integers, floating-point values can be either positive or negative.
- Java includes two floating-point types, which are float and double.

Float

- In Java, a value declared as float can hold a number in the range from around -3.402823 x 10 38 to around 3.402823 x 10 38.
- These types of values are also known as single-precision floating-point numbers and take up 32 bits (four bytes) of memory.
- You declare a single-precision floating-point number like this:

float identifier;

or

float identifier = value;

Double

- The second type of floating-point data, double, represents a double-precision value, which is a much more accurate representation of floating-point numbers.
- Because it allows for more decimal places.
- A double value can be in the range from -1.79769313486232 x 10 308 to 1.79769313486232 x 10 308 and is declared like this:

double identifier;

or

double identifier = value;

(3) Characters

- Often in your programs, you'll need a way to represent character values rather than just numbers.
- A character is a symbol that's used in text. The most obvious examples of characters are the letters of the alphabet, in both upper- and lowercase varieties.
- There are, however, many other characters, including not only things such as spaces, exclamation points, and commas, but also tabs, carriage returns, and line feeds.
- The symbols 0 through 9 are also characters when they're not being used in mathematical calculations.
- In order to provide storage for character values, Java features the char data type, which is 16 bits.
- However, the size of the char data type has little to do with the values it can hold. Basically, you can think of a char as being able to hold a single character.

- You declare a char value like this:

```
char c;
        or
char c = 'A';
```

(4) Boolean

- Many times in a program, you need a way to determine if a specific condition has been met.
- For example, you might need to know whether a part of your program executed properly.
- In such cases, you can use Boolean values, which are represented in Java by the boolean data type.
- Boolean values are unique in that they can be only one of two possible values: true or false.
- You declare a boolean value like this:

```
boolean identifier;
        or
boolean identifier = value;
```

- In the second example, value must be true or false. In an actual program, you might write something like this:

```
booleanfile_okay = true;
```

Ex :-

```
classdatatype
{
    public static void main(String args[])
    {
        // integer
        byte a = 127;
        short b = 32767;
        int c = 2147483647;
        long d = 1234567L;
        // floating points
        float e = 3.4f;
        double f = 0.000000987;
        // character type
        char g = 'a';
        // boolean type
        boolean h = a>b;

        System.out.println("------DATA TYPES------");
        System.out.println("byte: "+a);
        System.out.println("short: "+b);
        System.out.println("int: "+c);
        System.out.println("long: "+d);
        System.out.println("float: "+e);
        System.out.println("double: "+f);
        System.out.println("char: "+g);
        System.out.println("boolean: "+h);
    }
}
```

O/P→

```
byte: 127
short: 32767
int: 2147483647
long: 1234567
float: 3.4
double: 9.87E-7
char: a
boolean: false
```

Variables:

- The variable is the basic unit of storage in a Java program.
- A variable is defined by the combination of an identifier, a type, and an optional initializer.
- In addition, all variables have a scope, which defines their visibility, and a lifetime.

Declaring a Variable

- In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:
  type identifier [= value][, identifier [= value] ...] ;
- The type is one of Java's atomic types, or the name of a class or interface.
- The identifier is the name of the variable. You can initialize the variable by specifying an equal sign and a value.
- Keep in mind that the initialization expression must result in a value of the same type as thatspecified for the variable.
- To declare more than one variable of the specified type, use a commaseparated list.
- Here are several examples of variable declarations of various types. Note that some include an initialization
  .

```
int a, b, c;                    // declares three ints, a, b, and c.
int d = 3, e, f = 5;            // declares three more ints, initializing
                                // d and f.
byte z = 22;                    // initializes z.
double pi = 3.14159;            // declares an approximation of pi.
char x = 'x';                   // the variable x has the value 'x'.
```

Dynamic Initialization

- Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.
- For example, here is a short program that computes the length of the hypotenuse ofa right triangle given the lengths of its two opposing sides:

```
// Demonstrate dynamic initialization.

classDynInit
{
        public static void main(String args[])
```

```
        {
                double a = 3.0, b = 4.0;
                // c is dynamically initialized
                double c = Math.sqrt(a * a + b * b);
                System.out.println("Hypotenuse is " + c);
        }
}
```

Array in Java :

- An array is a structure that holds multiple values of the same type.
- The length of an array is established when the array is created (at runtime).
- After creation, an array is a fixed-length structure.
- An array element is one of the values within an array and is accessed by its position within the array.
- If you want to store data of different types in a single structure or if you need a structure Arrays are objects that contain a number of variables of the same type.
- These component variables are referenced using the integer indices 0,…,n-1, where nis the length of the array.
- The type of the array is identified by appending [] to the type of its components.
- For example, int[] identifies an array of type int, Object[] identifies an array of type Object, and char[][] identifies an array of type char.

Array Allocation

- When a variable of an array type is declared, the size of the array is not identified, and the array object is not allocated.
- To allocate storage for an array, you can use the new operator to create an array object of aspecific size. For example, the following statement:

```
Char ch[] = new char[24];
```

- Creates a char array of length 24, the individual component variables of which can be referenced by ch[0], ch[2], …, ch[23].
- The following statement creates an array of type Dice[] of length 6:

```
Dice[] d = new Dice[6];
```

- Arrays can also be allocated by specifying their initial values.
- For example, the following allocates a String array of length 7 that contains abbreviations for the days of the week:

```
String days[] = {"sun", "mon", "tue", "wed", "thu", "fri", "sat"};
```

- The length of an array can always be found by appending length to the name of the array.
- For example, days.length returns the integer 7 as the length of days[].
- Ex :- One Dim Array

```
classArr
{
        public static void main(String args[])
        {
                Int month_day[]={31,28,31,30,31,30,31,31,30,31,30,31};
                System.out.println("Days of April month are =  "+month_day[3]);
        }
}
```

Output :

Days of April month are = 30

❖Operators, Control Statements, Command Line Arguments

Operators in Java:

(1) Arithmetic Operators :

- The Java programming language supports various arithmetic operators for all floating-point and integer numbers.
- These operators are + (addition), - (subtraction), * (multiplication), /(division), and % (modulo).
- The following table summarizes the binary arithmetic operations in the Java programming language.

Operator	Use	Description
+	op1 + op2	Adds op1 and op2
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	Op1% op2	Computes remainder of dividing op1 by op2

Ex :-

```
classArithmaticOp
{
        public static void main(String args[])
        {
                int a=10,b=5;
                System.out.println("sum = "+(a+b));
                System.out.println("sub = "+(a-b));
                System.out.println("mul = "+(a*b));
                System.out.println("div = "+(a/b));
                System.out.println("mod = "+(a%b));
        }
}
```

Output :

```
sum = 15
mul = 5
sum = 50
mul = 2
mul = 0
```

(2) Relational Operator:

- A relational operator compares two values and determines therelationship between them.
- For example, != returns true if the two operands are unequal. This table summarizes the relational operators:

Operator	Use	Returns true if
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2

<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

Ex :-

```
classRelationalOp
{
        public static void main(String args[])
        {
                int a=5,b=10,c=5;
                System.out.println("a==b"+(a==b));
                System.out.println("a!=b"+(a!=b));
                System.out.println("a>b"+(a>b));
                System.out.println("a<b"+(a<b));
                System.out.println("a>=b"+(a>=b));
                System.out.println("a<=b"+(a<=b));
        }
}
```

Output :

```
a==b false
a!=b true
a>b false
a<b true
a>=b false
a<=b true
```

(3) Assignment Operator:

- The following table lists the shortcut assignment operators and their lengthy equivalents :

Operator	Use	Equivalent to
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

Ex :-

```
public class AssignmentOp
{
        public static void main (String[] args)
        {
                int a=1,b=10;
                a+=5;
                b=b+5;
                System.out.println(a);
                System.out.println(b);
        }
}
```

Output :
6
15

(4) Bitwise Logical Operator:

Operator	Use	Operation
&	op1 & op2	Bitwise and
|	op1 | op2	Bitwise or
^	op1 ^ op2	Bitwise xor
~	~op2	Bitwise complement

Ex :-Bitwise logical operator using One Dim Array

```
classBitwiseLogicalOp
{
        public static void main(String args[])
        {
        String binary[] = {
        "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
        "1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
        };
                int a = 3;
                int b = 6;
                int c = a | b;
                int d = a & b;
                int e = a ^ b;
                System.out.println(" a = " + binary[a]);
                System.out.println(" b = " + binary[b]);
                System.out.println(" a|b = " + binary[c]);
                System.out.println(" a&b = " + binary[d]);
                System.out.println(" a^b = " + binary[e]);
                }
}
```

Output :
a = 0011
b = 0110
a|b = 0111
a&b = 0010
a^b = 0101

(5) Conditional or Ternary operator:

- To find absolute (non-negative) value we used ternary operator.
- Relational operators often are used with conditional operators to construct more complex decision-making expressions.
- The **?:** operator is a conditional operator that is short-hand for an if-else statement:
  op1 ? op2 : op3;
  The ? : operator returns op2 if op1 is true or returns op3 if op1 is false.

Ex :-

```
public class TernaryOP
```

```
{
    public static void main(Stringargs[])
    {
            Int i, ans;
            i=-10;
            ans=i<0?-i:i;
        System.out.println("Absolute value of i = "+ans);
    }
}
```

Output :

Absolute value of i = 10

(6) Increment and Decrement Operator:

- The shortcut increment/decrement operators are summarized in the following table.

Operator	Use	Description
++	op++	Increments op by 1; evaluates to the value of op before it was incremented
++	++op	Increments op by 1; evaluates to the value of opafter it was incremented
--	op--	Decrements op by 1; evaluates to the value of op before it was decremented
--	--op	Decrements op by 1; evaluates to the value of op after it was decremented

Ex :- Post and Pre increment operator

```
classIncreDecreOp
{
 public static void main(String[] args)
 {
        int a=10,b=20;
        System.out.println("a="+a);
        System.out.println("b="+b);
        System.out.println("a++="+(a++));
        System.out.println("++b="+(++b));
        System.out.println("a="+a);
        System.out.println("b="+b);
 }
}
```

Output :

```
a=10
b=20
a++=10
++b=21
a=11
b=21
```

(7) Explain Bitwise Shift Operator:

- A shift operator performs bit manipulation on data by shifting the bits of its first operand right or left.
- This table summarizes the shift operators available in the Java programming language.

Operator	Use	Operation
>>	op1 >> op2	shift bits of op1 right by distance op2
<<	op1 << op2	shift bits of op1 left by distance op2
>>>	op1 >>> op2	shift bits of op1 right by distance op2(unsigned)

Ex :-

```
class Shift
{
        public static void main (String args[])
        {
                int x = 7;
                System.out.println("x = " + x);
                System.out.println("x >> 2 = " + (x >> 2));
                System.out.println("x << 1 = " + (x << 1));
                System.out.println("x >>> 1 = " + (x >>> 1));
        }
}
```

Output :

```
x = 7
x >> 2 = 1
x << 1 = 14
x >>> 1 = 3
```

Control Statements in Java:

Statement Type	Keyword
Looping	while, do-while , for
decision making	if-else, switch-case
Branching	break, continue, label:, return

(1) **If ... else**

- The If statement enables your program to selectively execute other statements, based on some criteria.
- This is the simplest version of the if…else statement: The block of statements will be executed if a condition is true.
- Generally, the simple form of if can be written like this:

```
if (expression)
{
        True statement
}
else
{
        False Statement
}
```

- What if you want to perform a different set of statements if the expression is false? You use the else.
- The else block is executed if the if part is false. Another form of the else statement.

```
classifelse
{
        public static void main(String arg[])
        {
                int a=10,b=20;
                if(a>b)
                {
                        System.out.println("a is greater than b");
                }
                else
                {
                        System.out.println("b is greater than a");
                }
        }
}
```

Output :

b is greater than a

(2) Switch ... case

- Statement to conditionally perform statements based on an integer expression.
- Following is a sample program, SwitchDemo, that declares an integer named month whose value supposedly represents the month in a date.
- The program displays the name of the month, based on the value of month, using the switch statement:

```
Class SwitchDemo
{
        public static void main(String[] args)
        {
                int month = 8;
                switch (month)
                {
                        case 1:  System.out.println("January"); break;
                        case 2:  System.out.println("February"); break;
                        case 3:  System.out.println("March"); break;
                        case 4:  System.out.println("April"); break;
                        case 5:  System.out.println("May"); break;
                        case 6:  System.out.println("June"); break;
                        case 7:  System.out.println("July"); break;
                        case 8:  System.out.println("August"); break;
                        case 9:  System.out.println("September"); break;
                        case 10: System.out.println("October"); break;
                        case 11: System.out.println("November"); break;
                        case 12: System.out.println("December"); break;
                        default: System.out.println("Hey, that's not a valid month!");
                        break;
```

```
                }
        }
}
```

Output: August

(3) While loop

- You use a while statement to continually execute a block of statements while a condition remains true.
- The general syntax of the while statement is:

```
while (expression)
{
        Statement;
}
```

- First, the while statement evaluates expression , which must return a boolean value.
- If the expression returns true, then the while statement executes the statement(s) associated with it.
- The while statement continues testing the expression and executing its block until the expression returns false.

Ex :-

```
public class While
{
        public static void main(String[] args)
        {
                int i=5;
                while(i>0)
                {

                        System.out.println("i="+i);
                        i--;
                }
        }
}
```

Output :

```
i=5
i=4
i=3
i=2
i=1
```

(4) do ... while loop

- The Java programming language provides another statement that is similar to the while statement, the do-while statement.
- The general syntax of the do-while is:

```
do
{
        statement(s);
} while (expression);
```

- Instead of evaluating the expression at the top of the loop, do-while evaluates the expression at the bottom.
- Thus the statements associated with a do-while are executed at least once.

```
class dowhile
{
    public static void main(String args[])
    {
            int i=5;
            do
            {
                    System.out.println("i="+i);
                    i--;
            }while(i>0);
    }
}
```

Output :

```
i=5
i=4
i=3
i=2
i=1
```

(5) for loop

- The for statement provides a compact way to iterate over a range of values.
- The general form of the for statement can be expressed like this:

```
for (initialization ; condition ; increment/decrement)
{
        Statements;
}
```

- The initialization is an expression that initializes the loop-it's executed once at the beginning of the loop.
- The conditional expression determines when to terminate the loop.
- This expression is evaluated at the top of each iteration of the loop.
- When the expression evaluates to false, the loop terminates.
- Finally, increment or decrement is an expression that gets invoked after each iteration through the loop.

Ex :-

```
public class ForLoop
{
    public static void main(String args[])
            {
                    for(int i=1;i<=5;i++)
                    {
                            System.out.println("i="+i);
                    }
     }
}
```

Output :

```
i=1
i=2
i=3
i=4
i=5
```

(6) Labeled Loop OR Outer Loop

- A label is an identifier placed before a statement.
- The label is followed by a colon (:).
- The labeled form of the **continue** statement skips the current iteration of an outer loop marked with the given label.

Ex :- Print only odd number between 1 to 10 using Label Loop or Outer Loop

```
class LabelLoop
{
    public static void main(String args[])
    {
            outer : for(int m=1;m<11;m++)
            {
                    if(m%2==0)
                    {
                            continue outer;

                    }
                    else
                    {
                            System.out.println(m);
                    }
            }

    }
}
```

(7) Using Break- Exit from a Loop

- By using break, you can force immediate termination of a loop, bypassing theconditional expression and any remaining code in the body of the loop.
- When a breakstatement is encountered inside a loop, the loop is terminated and program controlresumes at the next statement following the loop.

Ex :-

```
class BreakLoop
{
    public static void main(String args[])
    {
     for(int i=0; i<100; i++)
     {
            if(i == 6) break; // terminate loop if i is 6
            System.out.println("i: " + i);
```

```
        }
        System.out.println("Loop complete.");
      }
}
```

Output :

```
i: 0
i: 1
i: 2
i: 3
i: 4
i: 5
Loop complete.
```

(8) Continue statement

- You use the continue statement to skip the current iteration of a for, while , or do-while loop.
- The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop, basically skipping the remainder of this iteration of the loop.

Ex :-

```
class Continue
{
    public static void main(String args[])
    {
            for(int i=0;i<10;i++)
            {
                    System.out.print(i+" ");
                    if(i%2==0) continue;
                    System.out.println();
            }
    }
}
```

Output :

```
0 1
2 3
4 5
6 7
8 9
```

(9) Return statement

- The last of Java's branching statements is the return statement.
- You use return to exit from the current method.
- The flow of control returns to the statement that follows the original method call.
- The return statement has two forms: one that returns a value and one that doesn't.
- To return a value, simply put the value (or an expression that calculates the value)
  - after the return keyword:
- When a method is declared void, use the form of return that doesn't return a value.

Ex :-

```
class Return
```

```
{
    public static void main(String args[])
    {
            boolean t = true;
            System.out.println("Before the return.");
            if(t) return; // return to caller
            System.out.println("This won't execute.");
    }
}
```

Output :

Before the return.

Command Line Arguments:

- Sometimes you will want to pass information into a program when you run it.
- This is accomplished by passing command-line arguments to main( ).
- A command-line argument is the information that directly follows the program's name on the command line when it is executed.
- To access the command-line arguments inside a Java program is quite easy—they are stored as strings in a String array passed to the args parameter of main( ).
- The first command-line argument is stored at args[0], the second at args[1], and so on.
- For example, the following program displays all of the command-line arguments that it is called with:

```
// Display all command-line arguments.

class CommandLine
{
        public static void main(String args[])
        {
        for(int i=0; i<args.length; i++)
        System.out.println("args[" + i + "]: " +
        args[i]);
        }
}
```

- Try executing this program, as shown here:

```
Java CommandLine this is a test 100 -1
```

When you do, you will see the following output:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: test
args[4]: 100
args[5]: -1
```

❖Classes and Objects

Classes and Objects:
Classes:

What is Class?

- A class, in the context of Java, is templates that are used to create objects, and to define object data types and methods.
- Core properties include the data types and methods that may be used by the object. All class objects should have the basic class properties.
- Classes are categories, and objects are items within each category.
- Class can contain any of the following variable types.
  - **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
  - **Instance variables:** Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
  - **Class variables:** Class variables are variables declared with in a class, outside any method, with the static keyword.

The General Form of a Class:

```
class clsName
{
// instance variable declaration
type1 varName1 = value1;
type2 varName2 = value2;
:
:
typeN varNameN = valueN;
// Methods
rType1 methodName1(mParams1)
{
// body of method
}
:
:
rTypeN methodNameN(mParamsN)
{
// body of method
}
}
```

Example of a General Class

```
class clsSample
{
// Variables
```

```
static int ctr;
int i;
int j;

// Methods
int addition()
{
ctr++;
return i + j;
}
int NumberOfInstances()
{
return ctr;
}
}
```

Objects:

What is an Object?

- An Entity that has state and behavior is known as an object.
- For example, chair, bike, marker, pen, table, car etc.
- It can be physical or logical.
- The example of logical object is banking system.
- Object is an instance of a class. Class is a template or blueprint from which objects are created.
- An object has three characteristics:
    - **State:** represent data of an object. For example name of an object, colour etc.
    - **Behavior:** represent the behavior (functionality) of an object such as deposit, withdrawal etc..
    - **Identity:** object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Declaring Objects:

- You can create a class, you are creating a new data type. You can use this type of declare objects of that type.
- Obtaining of a class is a two step process.
- First you must declare a variable of the class type. Second you must acquire an actual, physical copy of the object and assign it to that variable.
- You can do this using the new operator. The new operator dynamically allocates memory for an object and returns a reference to it.
- Syntax to declare an object :

  Classname objectvariable = new classname ();

Example :

```
class objdecl
{
    class box
    {
            Int width, height, depth;
    }
```

```
    public static void main (String args[])
    {
        box  b = new box();
    }
}
```